

**Amendments to the Claims:**

The following listing of claims will replace all prior versions, and listings, of claims in the application:

1. (Currently Amended) A method for producing finite-state networks, comprising:
  - producing an input finite-state network having a set of paths, with at least one of the paths in the set of paths containing a delimited subpath; the delimited subpath encoding on an indicated side a delimited substring formatted as a regular expression;
  - creating a first temporary finite-state network by extracting from the first finite-state network the delimited subpath and eliminating the symbols on the indicated side of the delimited subpath;
  - creating a second temporary finite-state network by compiling the delimited substring formatted as a regular expression;
  - computing the cross-product of the first temporary finite-state network and the second temporary finite-state network to create a resulting finite-state network that is a compiled representation of the delimited substring; ~~and~~
  - producing an output finite-state network by replacing the delimited subpath in the input finite-state network with the resulting finite-state network; and
  - performing language processing using the output finite-state network.
2. (Original) The method according to claim 1, wherein the delimited substring is produced by concatenating the symbols along the indicated side of the delimited subpath.
3. (Original) The method according to claim 1, wherein each finite-state network is represented using a data structure of a computer program.

4. (Original) The method according to claim 1, wherein the input finite-state network, the output finite-state network, and the third temporary finite-state network are finite-state transducers.
5. (Original) The method according to claim 1, wherein the first temporary finite-state network and the second temporary finite-state network are simple finite-state automata.
6. (Original) The method according to claim 1, wherein the input finite-state network and the second temporary finite-state network are formed using a regular expression compiler.
7. (Original) The method according to claim 1, wherein the output finite-state network encodes a relation that involves a nonconcatenative process.
8. (Original) The method according to claim 7, wherein the nonconcatenative process is described by the regular expression in the delimited substring.
9. (Original) The method according to claim 8, wherein the nonconcatenative process described by the regular expression occurs in a natural language.
10. (Original) The method according to claim 7, wherein the nonconcatenative process is interdigitation.
11. (Original) The method according to claim 1, wherein the delimited substring is given by:  
  
     $\wedge [ X^n \wedge ]$ , which denotes a concatenation of  $n$  instances of  $X$ , where:  
  
     $\wedge [$  is a special character string that serves as an opening delimiter;  
  
     $\wedge ]$  is a special character string that serves as an closing delimiter;  
  
     $X$  is a language; and  
  
     $\wedge^n$  denotes a concatenation of  $n$  instances of  $X$ .

12. (Original) The method according to claim 1, further comprising providing the output finite-state network to execute an application routine.

13. (Previously Presented) The method according to claim 12, wherein the application routine is a morphological analyzer.

14. (Previously Presented) The method according to claim 12, wherein the application routine is a morphological generator.

15. (Currently Amended) A system for producing finite-state networks, comprising:

a regular expression compiler for producing an input finite-state network having a set of paths, with at least one of the paths in the set of paths containing a delimited subpath; the delimited subpath encoding on an indicated side a delimited substring formatted as a regular expression; ~~and~~

a compile-replace module coupled to the regular expression compiler for:

creating a first temporary finite-state network by extracting from the first finite-state network the delimited subpath and eliminating the symbols on the indicated side of the delimited subpath;

creating a second temporary finite-state network by compiling the delimited substring formatted as a regular expression with the regular expression compiler;

computing the cross-product of the first temporary finite-state network and the second temporary finite-state network to create a resulting finite-state network that is a compiled representation of the delimited substring; and

producing an output finite-state network by replacing the delimited subpath in the input finite-state network with the resulting finite-state network; and

a language processor that processes language using the output finite-state network.

16. (Original) The system according to claim 15, further comprising a runtime module that accesses the output finite-state network to execute an application routine.
17. (Original) The system according to claim 16, wherein the application routine performs morphological analysis.
18. (Original) The system according to claim 16, wherein the application routine performs morphological generation.
19. (Original) The system according to claim 15, wherein the delimited substring is given by:
- $\wedge [ X^n \wedge ]$ , which denotes a concatenation of  $n$  instances of  $X$ , where:
- $\wedge [$  is a special character string that serves as an opening delimiter;
- $\wedge ]$  is a special character string that serves as a closing delimiter;
- $X$  is a language; and
- $\wedge^n$  denotes a concatenation of  $n$  instances of  $X$ .
20. (Original) The system according to claim 15, further comprising a regular expression compiler for forming the input finite-state network and the second temporary finite-state network.